Chromatic Analysis of Numerical Program

David DEFOUR, LAMPS, Univ. of Perpignan
Franck Vedrine, Univ. Paris-Saclay CEA List
A few words about colors.... In RGB

Colors naturally provide visual information under additive property
Introduction

• Assessment
  • For some applications (DNN), we are more concerned by understanding the resulting value than by the propagation of errors

• Objective
  • Estimate the relations between input and output variables under additive property

• Solution
  • Use the concept of chromatic number to tint scalar or set of scalars
  • Each scalar is decomposed as the sum of tinted values
Chromatic number: Definition

- **A Chromatic Number** consists in associating a color to scalar or set of scalar in order to track them during computation
  - Corresponds to a triplet \( \langle x, k_x, V_x \rangle \)
    - \( x \) is the floating-point number
    - \( V_x \) is a vector of \( n \) floating-point numbers representing the weight of the \( n \) tints within \( x \) such that we have the **Additive property**
      - \( x = \frac{1}{k_x} \sum_{i=0}^{n} V_x[i] \)
  
- **Properties**
  - \( V_x \) Corresponds to a component-wise decomposition of numerical values
  - Multiple scalars can be set with the same tint (track multiple values at the same time and helps to reduce the dimensionality of the problem)
  - Need to set a “Garbage element” in \( V_x \) to collect contributions of non-chromatic numbers to preserve additive property.
    (Optional element if every computation were done without rounding error \( x = \sum_{i=0}^{n} V_x[i] \))
Chromatic number: Operations

• Set a new arithmetic on chromatic numbers:

  • Addition: \( \langle x, k_x, V_x \rangle + \langle y, k_y, V_y \rangle = \langle x + y, 1, \frac{V_x}{k_x} + \frac{V_y}{k_y} \rangle \)

  • Subtraction: \( \langle x, k_x, V_x \rangle - \langle y, k_y, V_y \rangle = \langle x - y, 1, \frac{V_x}{k_x} - \frac{V_y}{k_y} \rangle \)

  • Multiplication: \( \langle x, k_x, V_x \rangle . \langle y, k_y, V_y \rangle = \langle x \cdot y, k_x + k_y, y \cdot V_x + x \cdot V_y \rangle \)

  • Division: \( \frac{\langle x, k_x, V_x \rangle}{\langle y, k_y, V_y \rangle} = \langle \frac{x}{y}, \frac{V_x}{y^2} \rangle = \langle \frac{x}{y}, k_x + k_y, \frac{x}{y^2} \cdot V_y + \frac{V_x}{y} \rangle \)

  • Sqrt(x): \( \sqrt{\langle x, k_x, V_x \rangle} = \langle \sqrt{x}, 1, \frac{V_x}{k_x \sqrt{x}} \rangle \)

  • Any functions: \( f(\langle x, k_x, V_x \rangle, \langle y, k_y, V_y \rangle) = \langle f(x + y), k_x + k_y, \frac{f(x, V_y) + f(V_x, y)}{2} \rangle \)
Example 1: Cancellation

• Let consider the sequence of operations
  • $a = 2$
  • $b = 3$
  • $r = (a \cdot a) \cdot b - 12$

• In CA this corresponds to
  • $a = < 2, 1, [0,2,0]>$
  • $b = < 3, 1, [0,0,3]>$

  • $(a \cdot a) = < 4, 2, [0,8,0]>$
  • $(a \cdot a) \cdot b = < 12, 3, [0,24,12]>$
  • $(a \cdot a) \cdot b - 12 = < 0, 1, [-12, 8, 4]>$
Example 2: LP Digital Filter

- **Goal:**
  - Understand how output results are affected by input data, program parameters, etc over time
  - Relative weight in the output value of the input values
Example 3: Muller’s series

- Goal:
  - Give a fine interpretation of what is numerically happening in some pathological cases

\[
\begin{align*}
  u_0 &= 5.5 \\
  u_1 &= \frac{61.0}{11.0} \\
  u_{n+1} &= 111. - 1130. / u_n + 3000. / (u_n . u_{n-1})
\end{align*}
\]
Related works

1. Sensitivity analysis
   • Evaluate how variations in input parameters affect the output
   • Identify which input parameters have the greatest effect on the output
   • Issues:
     • Curse of dimensionality, inability to handle correlated input, difficult to interpret variation on multiple input

2. Automatic Differentiation
   • Compute the gradient at each step
   • Forward or backward according to the input/output dimensionality
   • Implementation:
     • Each number $X$ is replaced by a Dual Number $(x|x')$ where $x'$ is the derivative such that $X = x + x'\varepsilon$ with $\varepsilon$ an abstract number such that $\varepsilon^2 = 0$.
     • Problematic with complex multivariate program as untracked variables are not taken into account
       (Solution: rely on automated sparsity detection of the Jacobian matrix)
Graphical illustration of CA vs AD

Relative weight of x (red) and y (blue) in f(x,y)

\[ f(x, y) = x^2 + y^2 \]

Local steepness measure

Chromatic analysis

AD analysis

Gradient's vector field for f(x,y)

Grad Norm

10/13
Experiments: inference in DNN MNIST

1 pixel = 1 index

784,100,50,10 Network

Possible usage: adversarial attack to alter output probability classification by minimizing the number of modified pixel
Execution overhead: x100 on time, x20 on memory
Experiments: Training DNN MNIST

• Goal
  • Track the weight of image class during learning phase
  • Understand the network’s numerical behavior

• Methodology
  • Tint according to its image classification (0 to 9)
  • Each of the 84k coefficients are decomposed according to image classification

• Output
  • Resulting network made of coefficient decomposed as chromatic numbers tinted according to the input images class

• Overhead
  • x1800 time (not possible to use optimized BLASS)
  • x12 on memory

For an image:
  each pixel = same index
Conclusion

• Chromatic analysis
  • Additive decomposition of results according to tinted values/set of values
  • Allows fusion of data to limit the dimensionality problem encountered with other analysis
  • Helps understand what is important among input values, constant, scalar and their numerical relation (ex: cancellations)
  • Preserve the input data structures

• Future works
  • Thanks to the additive property, it is possible to combine this analyse with an iterative refinement algorithm to reduce the memory overhead
    • Start with a few sets of tracked values (low memory / low computational overhead),
    • Restart the analyses by splitting only sets of variables which are having a large impact
  • Combine chromatic analysis with others to reduce the cost of global sensitivity analysis
    • Will help focusing on variables of interest.
  • Investigate various tinting mechanism
    • According to data type, time, location (functions, MPI Process...)
  • Test on real life program (numerically instruct abnormality)
Example: Error Free Transformation

\[(s, t) = \text{Fast2Sum}(a, b) \quad \text{// } a \gg b\]
\[s = a + b\]
\[r = s - a\]
\[t = b - r\]

\[a = \langle a, 1, [0, a, 0]\rangle\]
\[b = \langle b, 1, [0, 0, b]\rangle\]

\[s = \langle a, 1, [0, a, b]\rangle\]
\[r = \langle 0, 1, [0, 0, b]\rangle\]
\[t = \langle b, 1, [0, 0, 0]\rangle\]

\[0 \neq 0 + 0 + b\]

Does not take into account rounding error
Example: Error Free Transformation

\[(s, t) = \text{Fast2Sum}(a, b) \quad // \quad a \gg b\]
\[s = a + b\]
\[r = s - a\]
\[t = b - r\]

\[a = \langle a, 1, [0, a, 0] \rangle\]
\[b = \langle b, 1, [0, 0, b] \rangle\]
\[s = \langle a, 1, [0, a, b] \rangle\]
\[r = \langle 0, 1, [0, 0, b] \rangle\]
\[t = \langle b, 1, [0, 0, 0] \rangle\]

\[a = \langle a, 1, [0, 0, a, 0] \rangle\]
\[b = \langle b, 1, [0, 0, 0, b] \rangle\]
\[s = \langle a, 1, [0, -b, a, b] \rangle\]
\[r = \langle 0, 1, [0, -b, 0, b] \rangle\]
\[t = \langle b, 1, [0, b, 0, 0] \rangle\]

Does not take into account rounding error
=> Include a rounding error term
=> Dedicated routines for EFT
Chromatic number: Implementation

• Space and time complexity grows linearly with the number of tinted values.
  • Example: A chromatic analysis on a 8 Mb dense matrix will lead to 8 Tb of intermediate representation.
  • C++/Python implementation with $V_x$ stored either as a dense/sparse structure (vector/dictionary)

• Optimization: Fusion of small contributions
  • Discard tinted element which are becoming too small compared to others and accumulate them in the garbage element ($\frac{|V_x[i]|}{V_{x,i}} \geq C$ with $C$ a tunable parameter typically set to $2^{53}$ for double precision). Particularly useful when used when $V_x$ is a dictionary structure.

• Optimization: Error element
  • Set an element to track rounding errors performed on $x$ in $\langle x | V_x \rangle$
  • Accumulate rounding error similarly to compensated algorithm (use of EFT & extended precision)
Chromatic number: Implementation

- Optimization: Refinement algorithm
  - Start the chromatic analysis by aggregating the maximum number of value under the same tint in order to minimize the size of $V_x$.
  - Detect which tint account for the most and restart the computation by subdividing the selected tint, while detecting under-approximation (cancellation within a tint)

Algorithm 1 Contribution refinement subdivision algorithm

```plaintext
Require: O=func(I) the function to analyse
Require: I the set of scalar to track
Require: card(I) = N, and O = \langle o, V_o \rangle
I' = split(I)
\rightarrow Initial Splitting

O' = func(I')
S = False
for i in I' do
  if |o'| > k_0 |V_o'[1]| and |V_o'[i + 2]| > k_1 |V_o'[1]| and card(I'[i]) > 1 then
    I' = split(I'[i])
    S = True
  end if
end for
while S
```
Experiment: Sparse solver

- **Matrix from MatrixMarket:**
  - BCSSTK13: size 2003 x 2003; 42943 entries; estimated conditioned number $4.6 \times 10^{10}$
  - BCSSTK14: size 1806 x 1806; 32630 entries; estimated conditioned number $1.3 \times 10^{10}$

- **Execution time in sec. and memory to solve BCSSTK14 between Python and C++ version.**
  - 6-10x overhead in Python, 10-700x overhead in C++ (due to the sparsity of the system)
  - Memory usage grows linearly => x500-1000 on memory for 1000 tinted values

<table>
<thead>
<tr>
<th>Number of tinted value followed</th>
<th>no-instr</th>
<th>1</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>250s /70Mo</td>
<td>1634s /108Mo</td>
<td>2022s /125Mo</td>
<td>2500s /156Mo</td>
</tr>
<tr>
<td>C++</td>
<td>0.13s /21Mo</td>
<td>1.3s /26Mo</td>
<td>40s /135Mo</td>
<td>92s /253Mo</td>
</tr>
</tbody>
</table>
Experiment N°3: Sparse solver

- Iterative refinement algorithm, starting with a 4x4 subdivision according to the index in each direction of the matrix BCSSTK13.
- Stops after 5 iterations in 836 sec.

Reference
Analysis conducted while keeping the 128 most contributing tint in each cell. (2205 sec)
=> More time consuming and less precise than the iterative algorithm
import numpy as np
from scipy.signal import butter, lfilter, freqz
import matplotlib.pyplot as plt

# Create a low-pass Butterworth filter
def butter_lowpass(cutoff, fs, order=5):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    return b, a

# Apply the filter to the input signal
def butter_lowpass_filter(data, cutoff, fs, order=5):
    b, a = butter_lowpass(cutoff, fs, order=order)
    y = lfilter(b, a, data)
    return y

# Example usage
# Generate some random input data
fs = 100.0  # Sample rate (Hz)
t = np.linspace(0, 1, int(fs), endpoint=False)
data = np.sin(2 * np.pi * 5 * t) + 0.5 * np.sin(2 * np.pi * 20 * t)

# Filter parameters
order = 6
f_c = 10.0  # Desired cutoff frequency (Hz)

# Apply the filter to the input data
filtered_data = butter_lowpass_filter(data, f_c, fs, order)